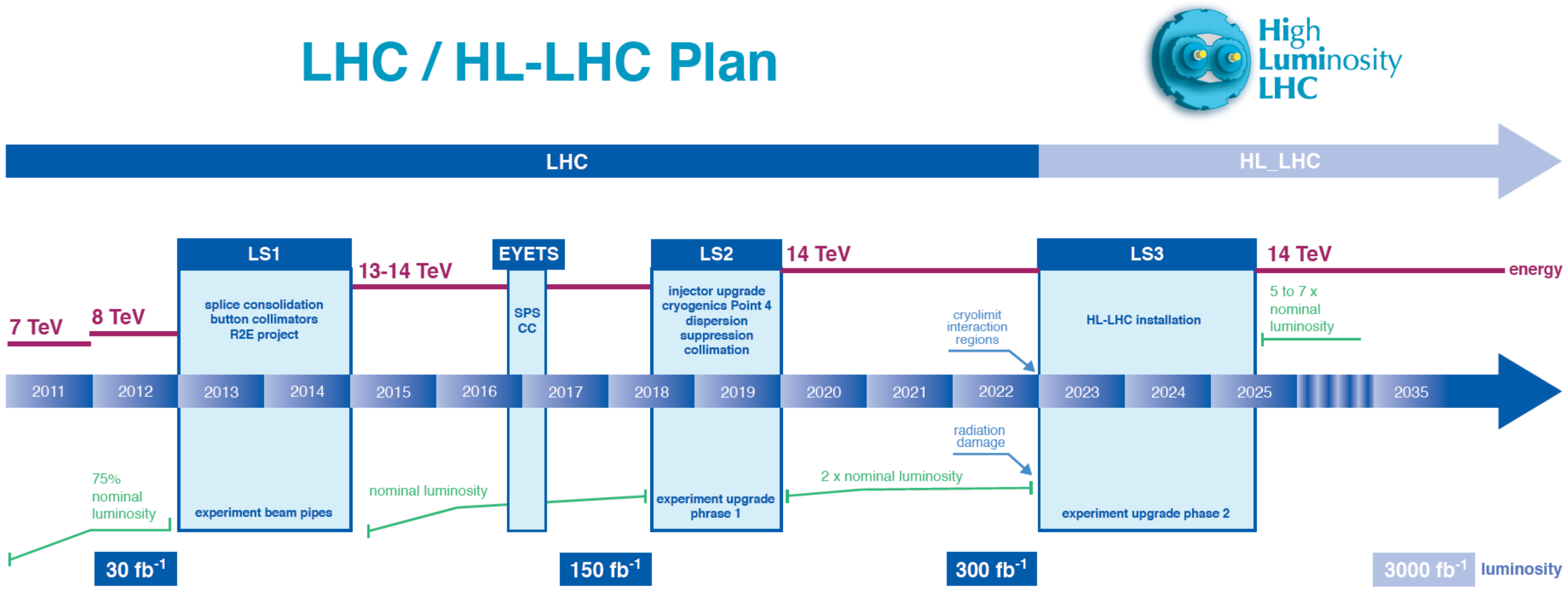
The next Heavy Ion Jet INteraction Generator HIJING++

G.G. Barnaföldi[1], W.T. Deng[7], M. Gyulassy[5] Sz.M. Harangozó[1,6], G.Y. Ma[2,3], G. Papp[2,3], O. Nieberl[6], X-N. Wang[2,3,4], B.W. Zhang[2,3]

1、 Wigner Research Centre for Physics, Hungarian Academy of Sciences
2、 Institute of Partical Physics, Central China Normal University
3、 Key Laboratory of Quark & Lepton Physics, China
4、 Lawrence Berkeley National Laboratory
5、 Columbia University in the City of New York.
6、 Department of Theoretical Physics, Eötvös Loránd University
7、 Huazhong University of Science and Technology.

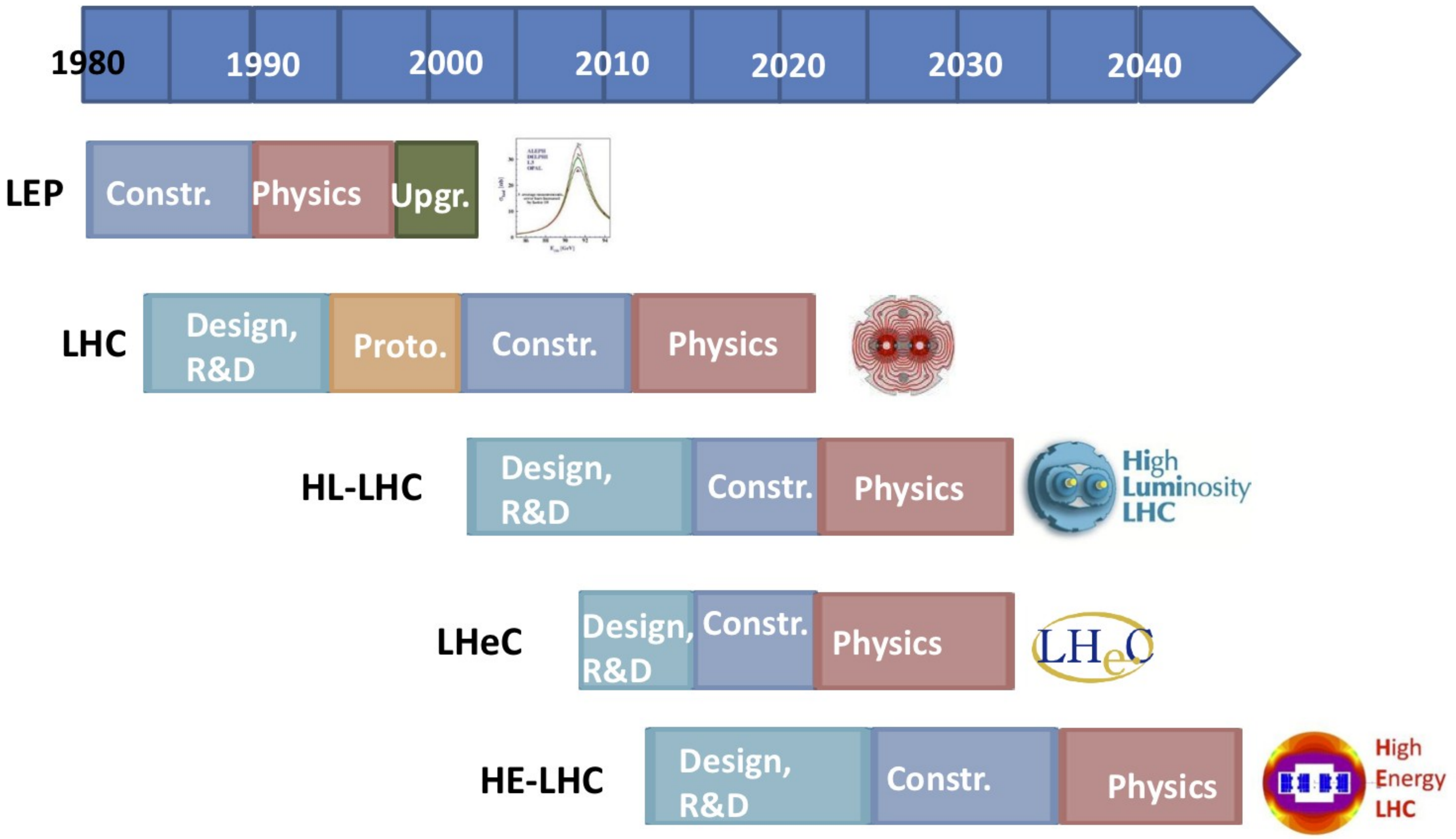
Motivation

- Need for more data, higher statistics….



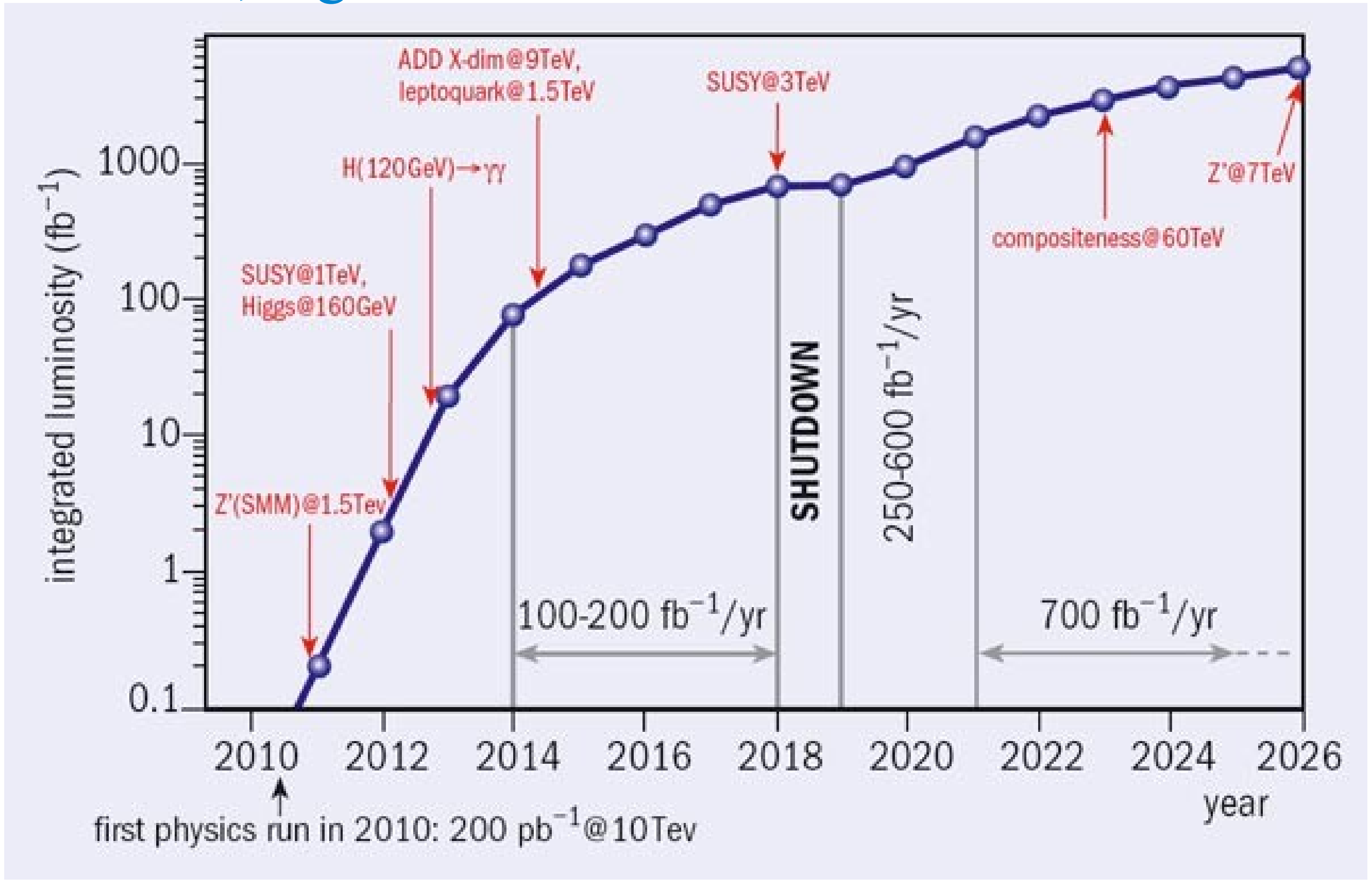
Motivation

- Need for more data, higher statistics….



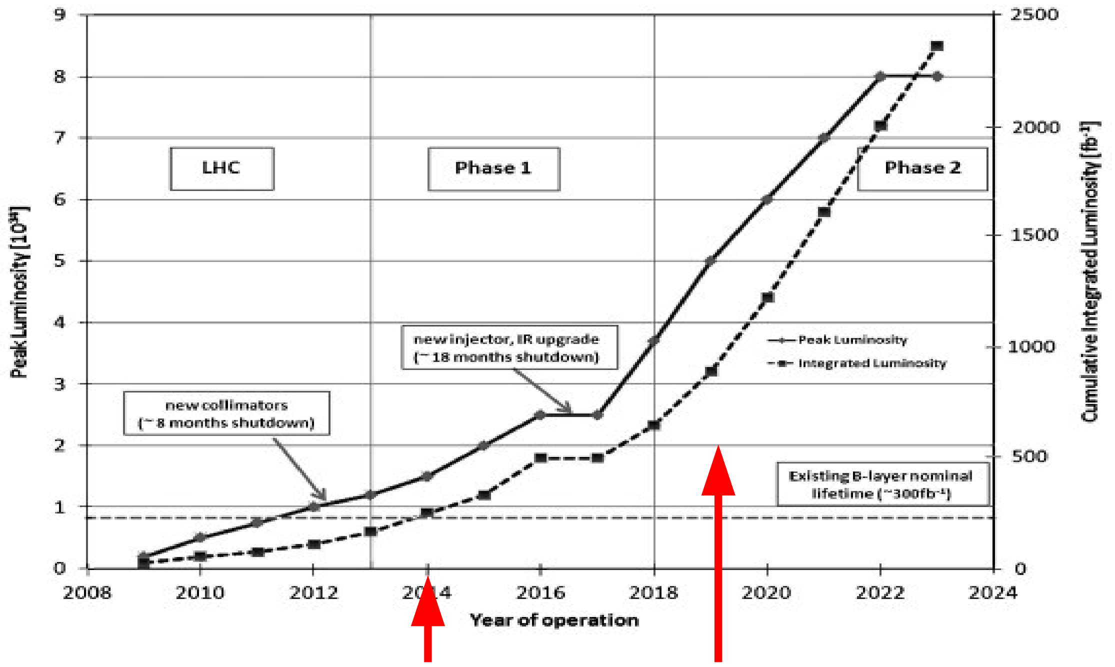
Motivation

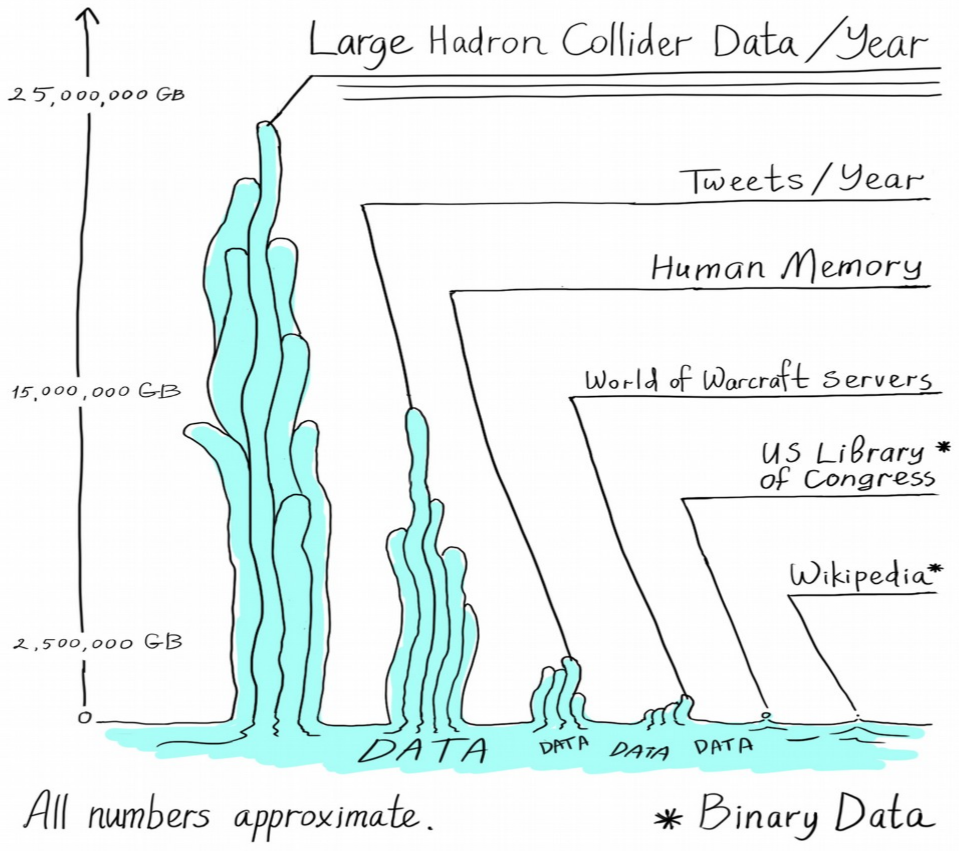
- Need for more data, higher statistics….



Motivation

- WLCG – Worldwide LHC Computing GRID:
  - 15-20 Petabytes data per year
  - ...and more after LHC upgrades

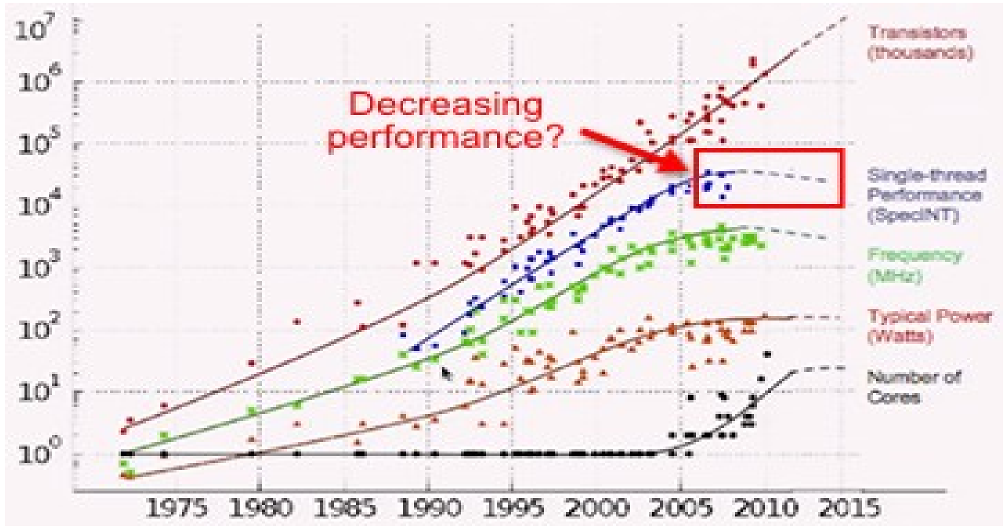




Fast computing

- Moore's law:

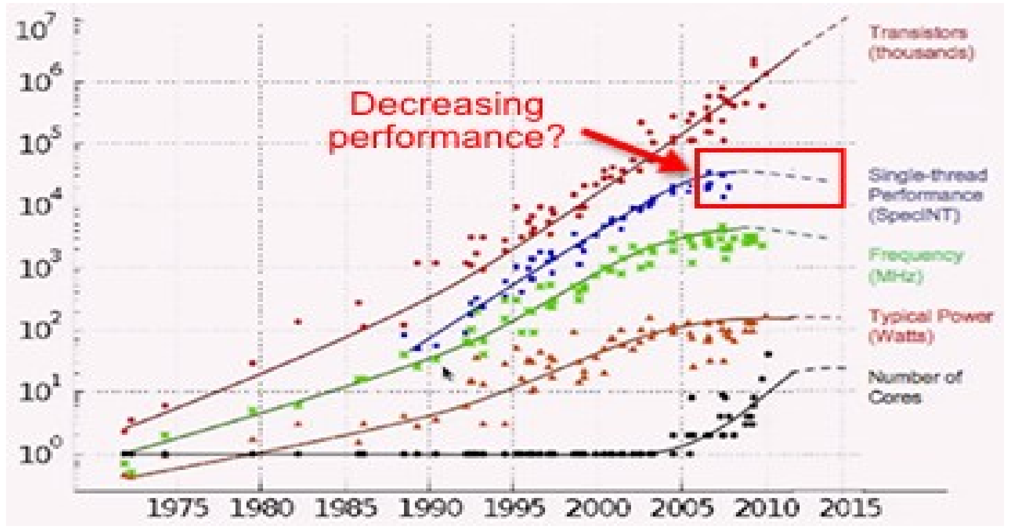
Every 2nd year the number of transistors (integrated circuits) are doubled in computing hardwares.



Fast computing = parallel computing

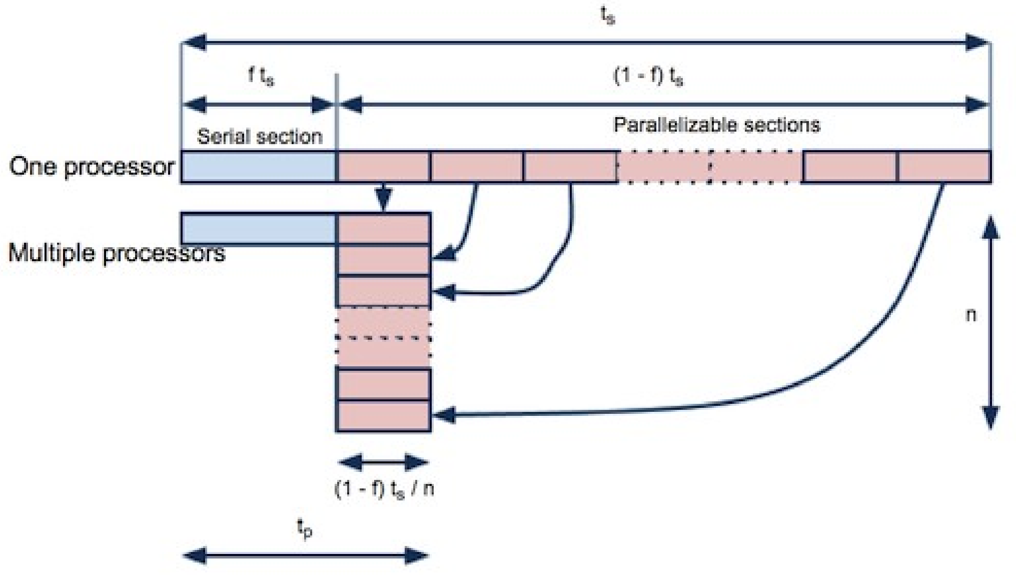
- Moore's law:

Every 2nd year the number of transistors (integrated circuits) are doubled in computing hardwares.



- Amdalh's law:

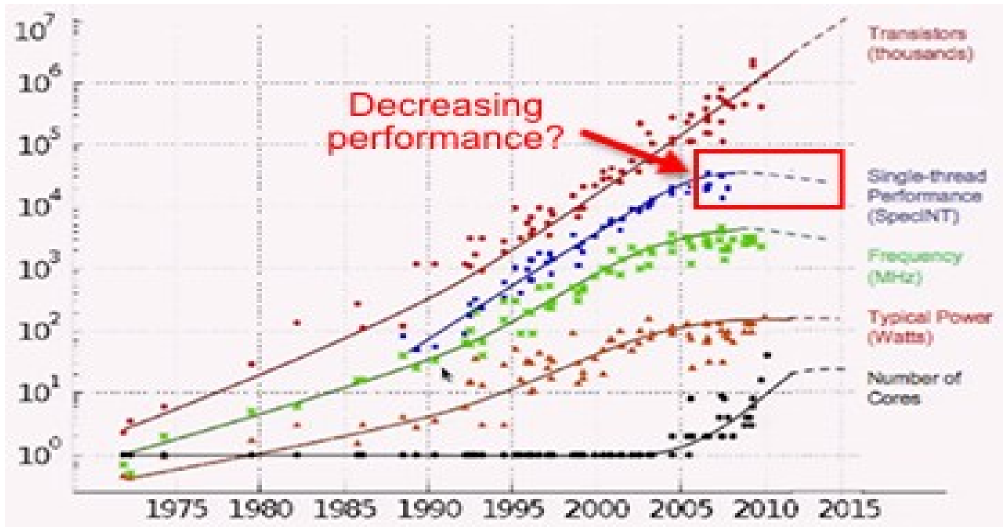
The theoretical speedup is given by the portion of parallelizable program, p, & number of processors (threads), N, is:



Fast computing = parallel computing

- Moore's law:

Every 2nd year the number of transistors (integrated circuits) are doubled in computing hardwares.



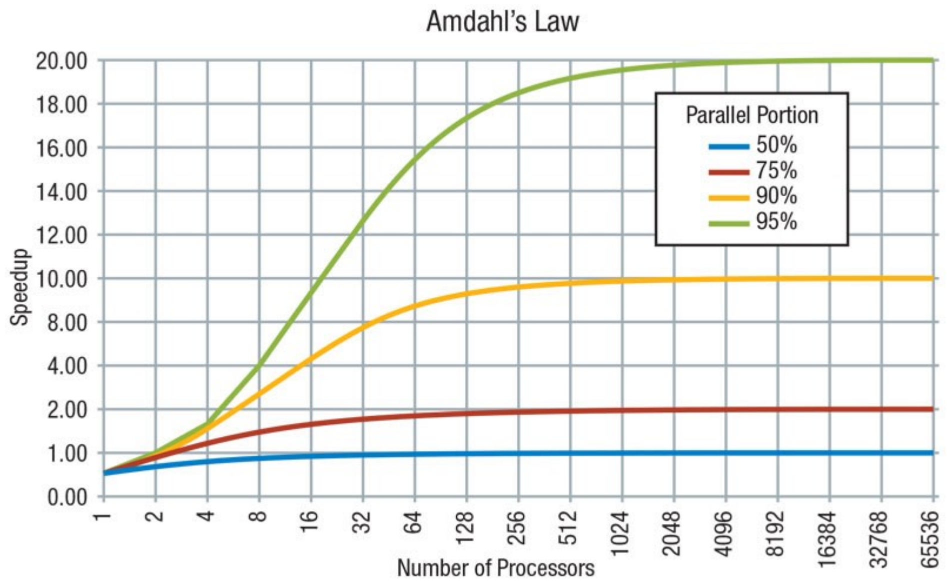
- Amdalh's law:

The theoretical speedup is given by the portion of parallelizable program, p, & number of processors (threads), N, is:

$$\text{Speedup}(N) = \frac{1}{(1-P)+\frac{P}{N}}$$

Serial part of job = 1 (100%) - Parallel part

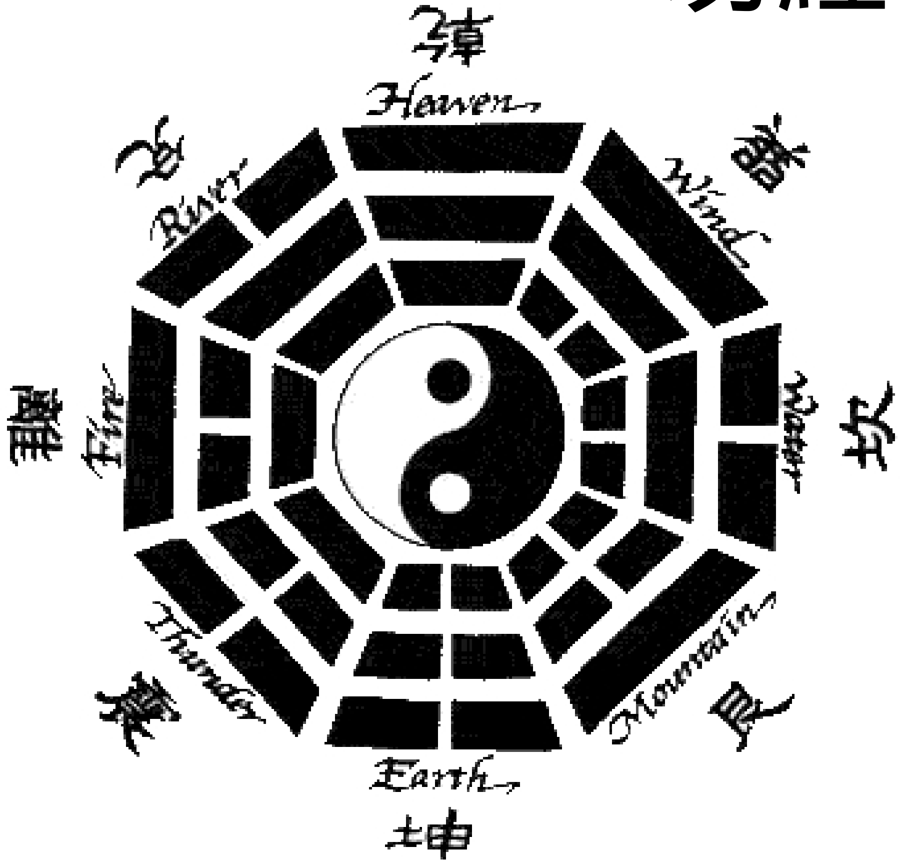
Parallel part is divided up by N workers



Introduction to HIJING++

- HIJING(Heavy-Ion Jet INteraction Generator)

易經



Bagua (eight simbols)

fundamental principles of reality

adjoint representation 8 of *SU(3)*

Historical Review

- HIJING(Heavy-Ion Jet INteraction Generator)

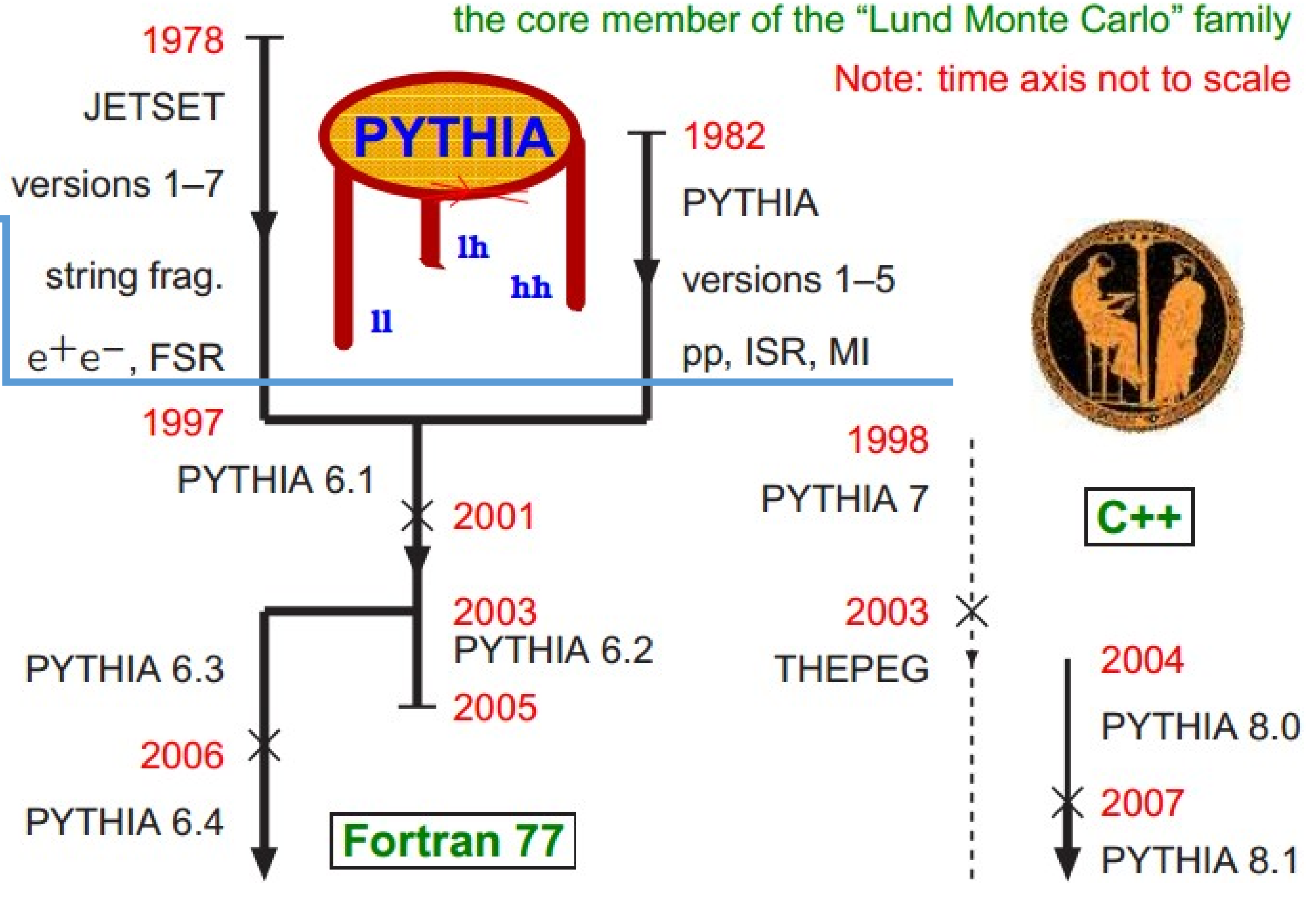
- HIJING versions
  - FORTRAN v1.36, v2.553
  - C++ v3.0

Reasons to use C++

Object oriented language: Hierarchy, Modularity

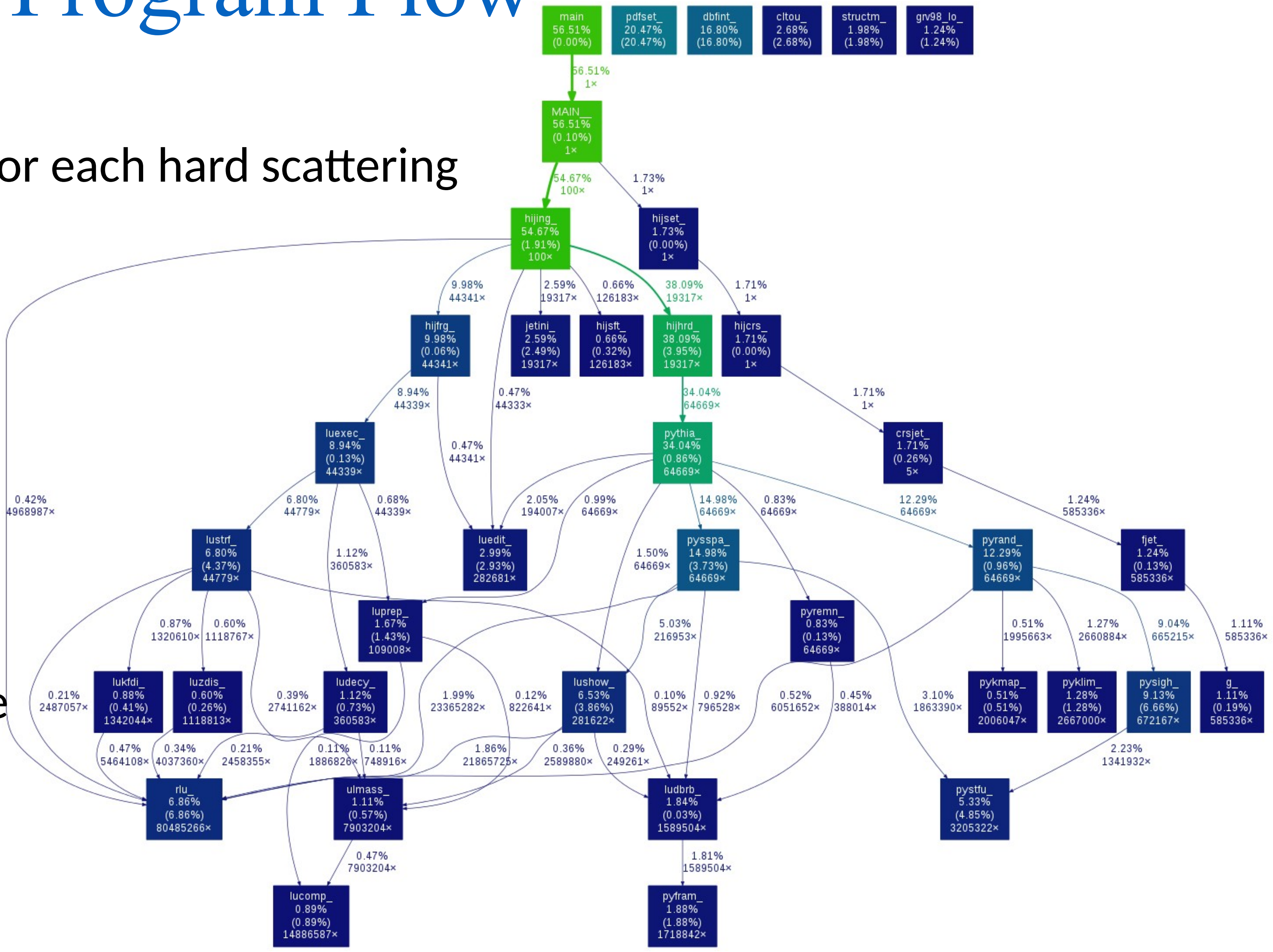
C++11/14 has thread support
and compatibility with OpenCL

PYTHIA history



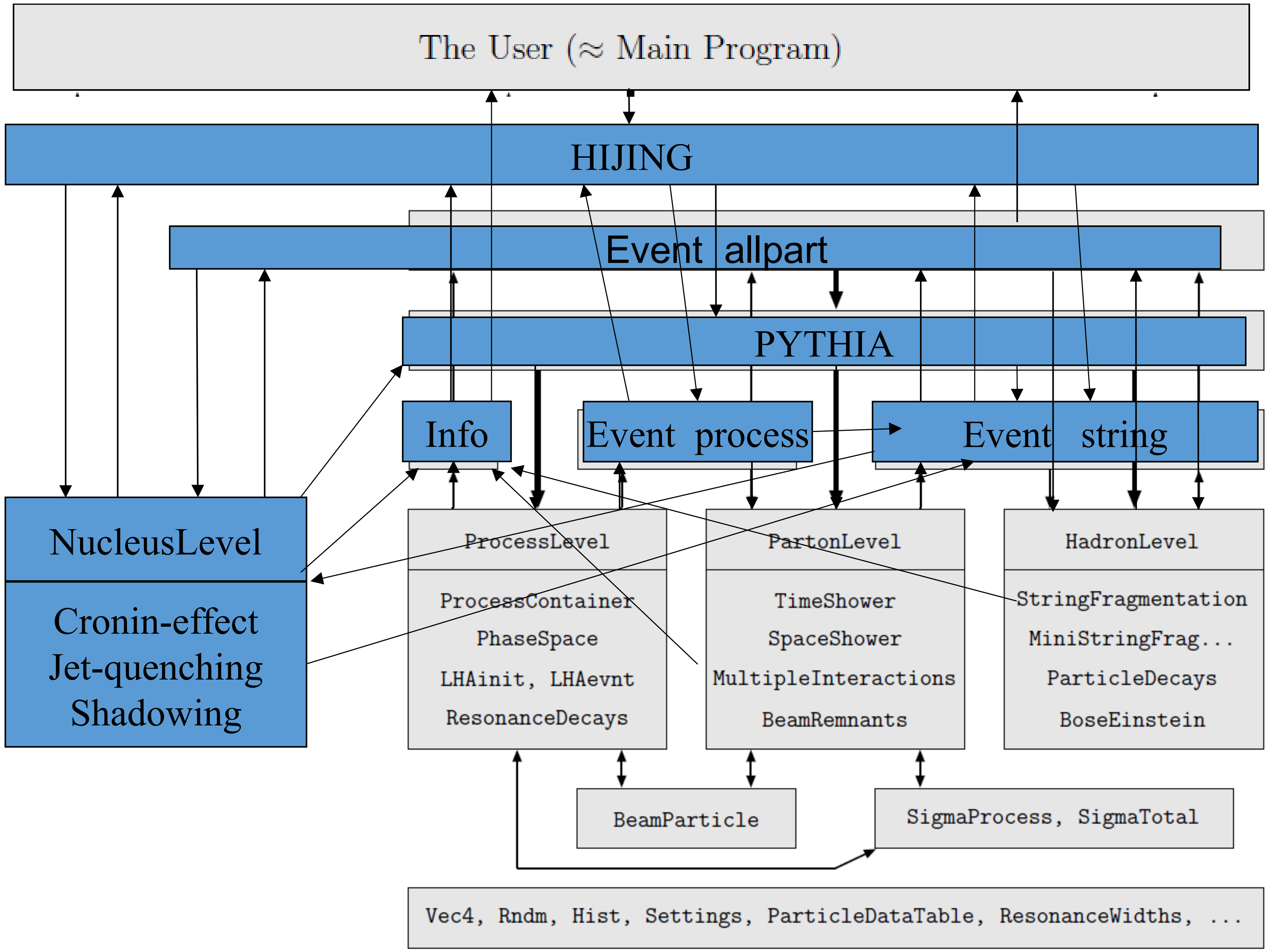
Program Flow

- Generation of kinetic variables for each hard scattering with Pythia 5.3
- Multiple soft gluon exchanges between valence- and diquarks
- String hadronization according to Lund fragmentation scheme


main_ 56.51% (0.00%)
pdfset_ 20.47% (20.47%)
dbfint_ 16.80% (16.80%)
cltou_ 2.68% (2.68%)
structm_ 1.98% (1.98%)
grv98_lo_ 1.24% (1.24%)
MAIN__ 56.51% (0.10%) 1×
hijing_ 54.67% (1.91%) 100×
hijset_ 1.73% (0.00%) 1×
hijfrg_ 9.98% (0.06%) 44341×
jetini_ 2.59% (2.49%) 19317×
hijsft_ 0.66% (0.32%) 126183×
hijhrd_ 38.09% (3.95%) 19317×
hijcrs_ 1.71% (0.00%) 1×
luexec_ 8.94% (0.13%) 44339×
pythia_ 34.04% (0.86%) 64669×
crsjet_ 1.71% (0.26%) 5×
lustrf_ 6.80% (4.37%) 44779×
luedit_ 2.99% (2.93%) 282681×
pysspa_ 14.98% (3.73%) 64669×
pyrand_ 12.29% (0.96%) 64669×
fjet_ 1.24% (0.13%) 585336×
luprep_ 1.67% (1.43%) 109008×
pyremn_ 0.83% (0.13%) 64669×
lukfdi_ 0.88% (0.41%) 1342044×
luzdis_ 0.60% (0.26%) 1118813×
ludecy_ 1.12% (0.73%) 360583×
lushow_ 6.53% (3.86%) 281622×
pykmap_ 0.51% (0.51%) 2006047×
pyklim_ 1.28% (1.28%) 2667000×
pysigh_ 9.13% (6.66%) 672167×
g_ 1.11% (0.19%) 585336×
rlu_ 6.86% (6.86%) 80485266×
ulmass_ 1.11% (0.57%) 7903204×
ludbrb_ 1.84% (0.03%) 1589504×
pystfu_ 5.33% (4.85%) 3205322×
lucomp_ 0.89% (0.89%) 14886587×
pyfram_ 1.88% (1.88%) 1718842×

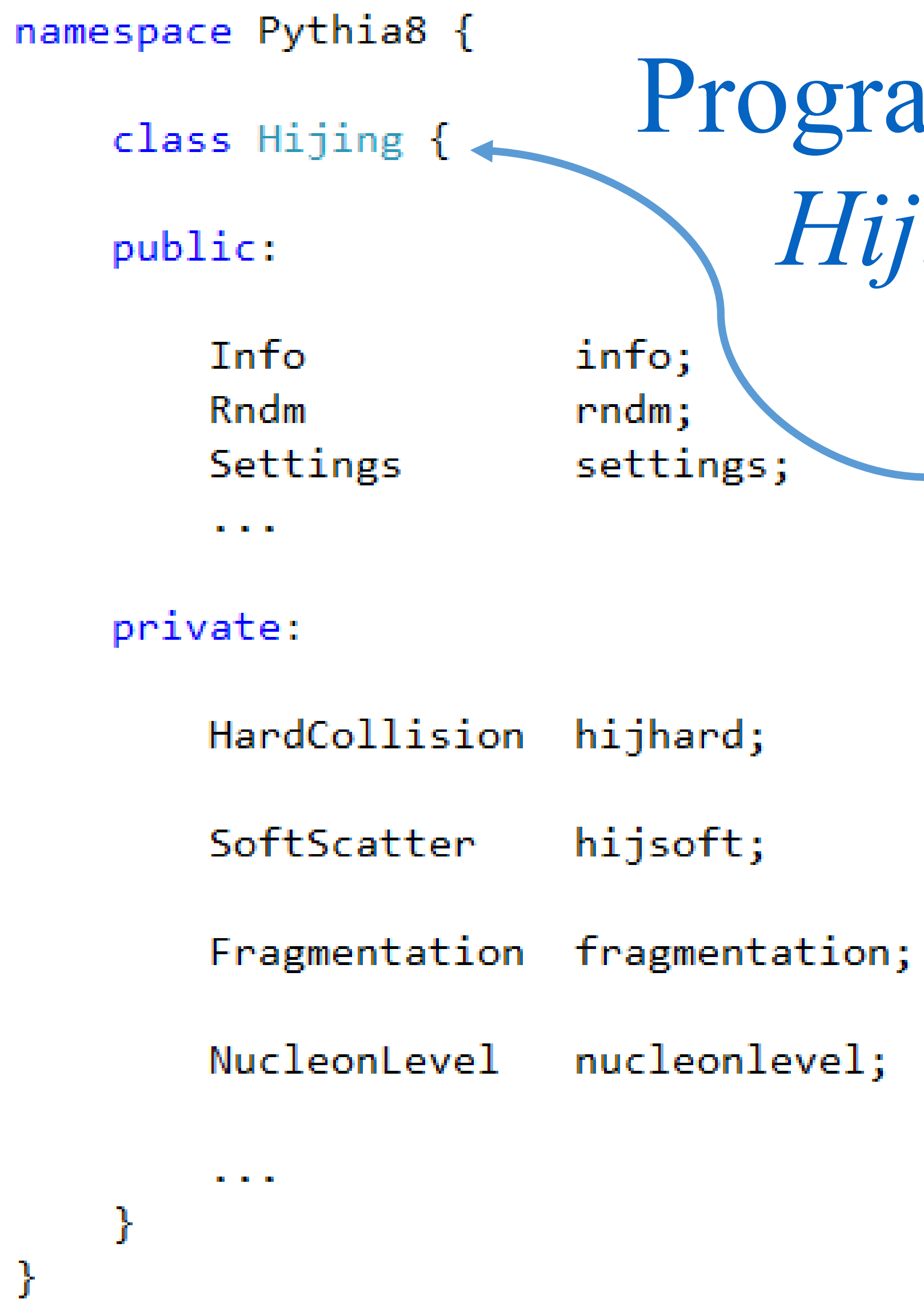
Program Structure

- Pythia8 namespace containers
- Structure similarities
- Actual program flow is more complicated



Program Structure

Hijing class



```
namespace Pythia8 {

    class Hijing {

    public:

        Info            info;
        Rndm            rndm;
        Settings        settings;
        ...

    private:

        HardCollision   hijhard;

        SoftScatter     hijsoft;

        Fragmentation   fragmentation;

        NucleonLevel    nucleonlevel;


        ...
    }
}
```

- Processes ordered in class hierarchy
- Former common blocks → class variables
- Processes called through object functions

// Class for handling the hard collisions

// Class for handling the soft interactions

// Class for handling the Lund string fragmentation

// Class for the nuclear effects

Main example

Usual form kept for regular users

FORTRAN

```
PROGRAM TEST
...
PARM(1) = 'DEFAULT'
VALUE(1) = 80060
CALL PDFSET(PARM, VALUE)
CALL GetDesc()
...

CALL HIJSET(EFRM, FRAME, PROJ, TARG, IAP, IZP, IAT, IZT)

N_EVENT=1E6
DO 200 IE = 1, N_EVENT
    CALL HIJING(FRAME, BMIN, BMAX)
200 CONTINUE

STOP
END
```

Form also similar to Pythia 8.x

C++

```
#include "Hijing.h"

using namespace Pythia8;

int main() {
    Hijing hijing("../xmldoc", true);
    hijing.readString("PDF:pSet = LHAPDF6:GRV98lo");

    bool okay = hijing.init(200.0, frame,
                  "A", "A", 197, 79, 197, 79);
    if (!okay) return 1;

    int MaxEvent = 1e6;
    for (int iEvent = 0; iEvent < MaxEvent; ++iEvent)
        hijing.next(frame, 0.0, 0.0);
}
```

Program Features

- Calculation by improved models
- Pythia like prompt Histogram creation
- CPU level Parallel computing

```
const std::size_t num_threads = std::thread::hardware_concurrency();
for (std::size_t i = 0u; i < num_threads; ++i){
    async_hijing.at(i) = std::unique_ptr<Hijing>(new Hijing);
}
for (std::size_t I = 0; I < num_threads; ++I){
    ...async run...
    okay[I] = async_hijing[I]->init(...);
    for (int iEvent = 0; iEvent < numEvent; ++iEvent)
        async_hijing[I]->next(...);
    for (int i = 0; i < async_hijing[I]->event.size(); ++i)
        if(...) hist[I]->fill(...);
}
```

THE C++ STANDARD TEMPLATE LIBRARY

- AliRoot compatibility (planned)

Model Improvements

- Shadowing

  HIJING 2.0 fits RHIC data well
  Improvements are needed for LHC energy

  $$R_i(x,b) \;\rightarrow\; R_i(Q,x,b)$$

- Jet-Quenching

  Various models: accuracy ⊒ ⊓ speed

- Soft QCD radiation

  updated ARIADNE calls

- (already implemented improvements since v1.36)

Dependencies & External packages

- Boost

```
sudo apt-get install libboost-all-dev
```

- LHAPDF 6

```
./configure –prefix=$HOME/…/share/LHAPDF
make all
```

insert downloaded PDF library to $HOME/…/share/LHAPDF

optionally modify **pdfsets.index**, add set if needed

```
export LD LIBRARY PATH=<library path>
```

- Pythia 8

```
./configure --with-lhapdf6-lib=$HOME/…/lib \
--with-boost-lib=/usr/lib/x86_64-linux-gnu
make –j4
```

- GSL (optional)

HIJING **make** option

Data Analysis

```
#include "Hijing.h"
using namespace Pythia8;

int main() {
    Hist dndpT("dn/dpT for charged particles", 100, 0., 10.);
    ofstream ch_file("ch_hist.dat");
    ...
    bool okay = hijing.init(efrm, frame, proj, targ,
                            aproj, zproj, atarg, ztarg);
    if (!okay) return 1;
    int MaxEvent = 1e6;
    for (int iEvent = 0; iEvent < MaxEvent; ++iEvent) {
        hijing.next(frame, bmin, bmax);
    for (int i = 0; i < hijing.event.size(); ++i)
    if (hijing.event[i].isFinal() && hijing.event[i].isCharged())
        dndpT.fill(hijing.event[i].pT());
    }
    dndpT *= 1.0 / MaxEvent;
    cout << dndpT;
    dndpT.table(ch_file);
    ...
    return 0;
}
```

Pythia 8 Histogram class available

Selection has to be made for every particle

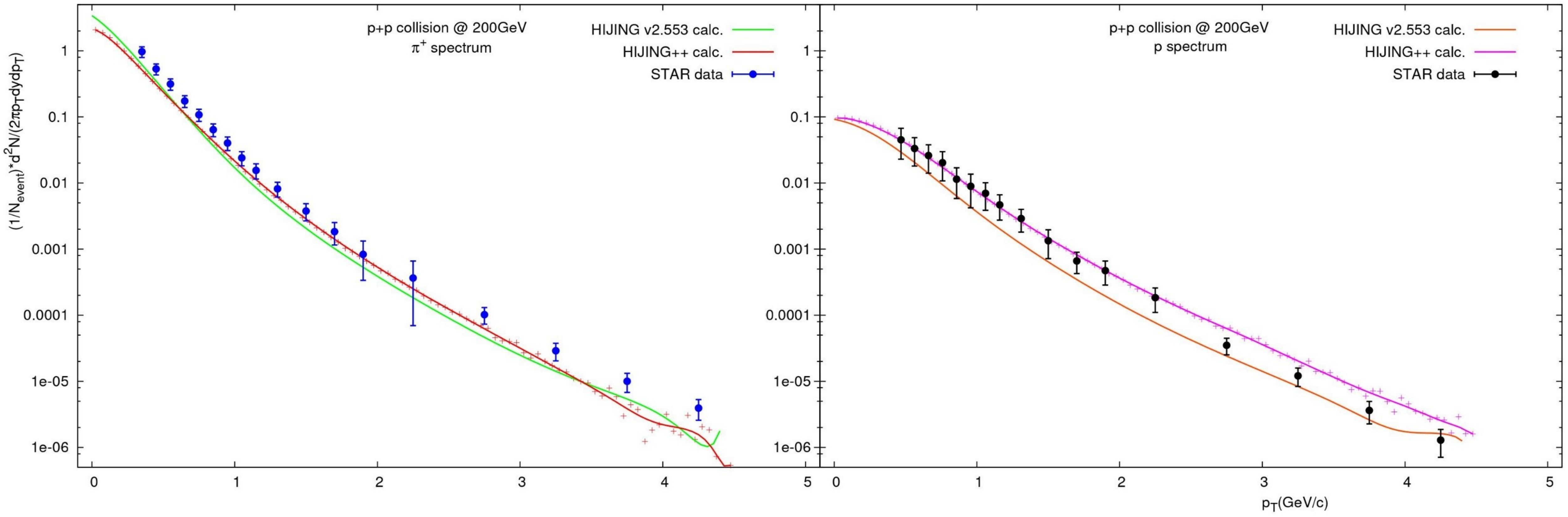
Hist::fill(double Input);

Normalization

standard output and file output both provided

Apetizer plots for the RHIC era

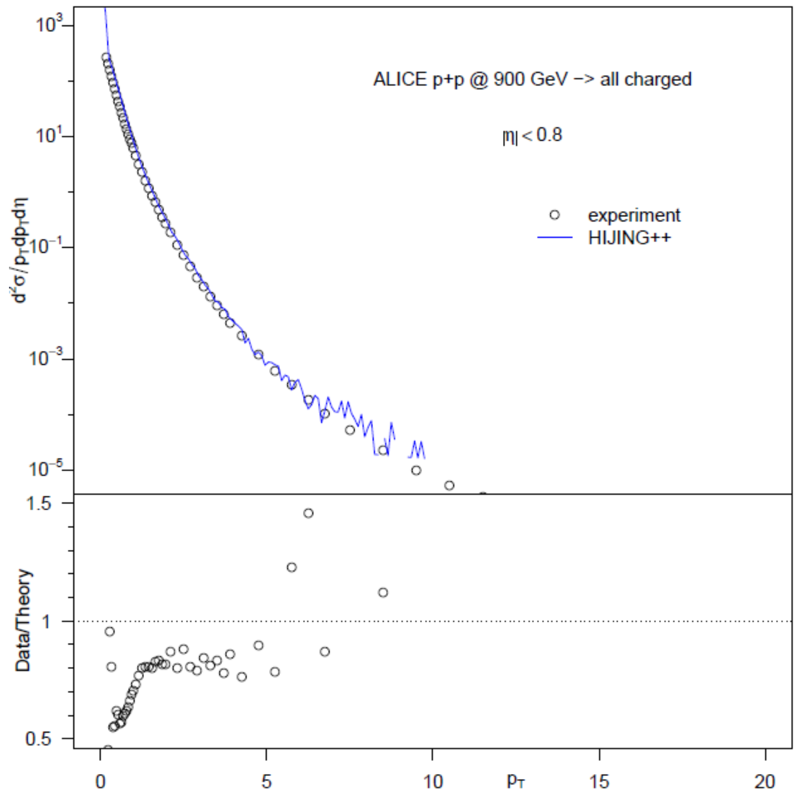
Code validation with „old” version and RHIC data

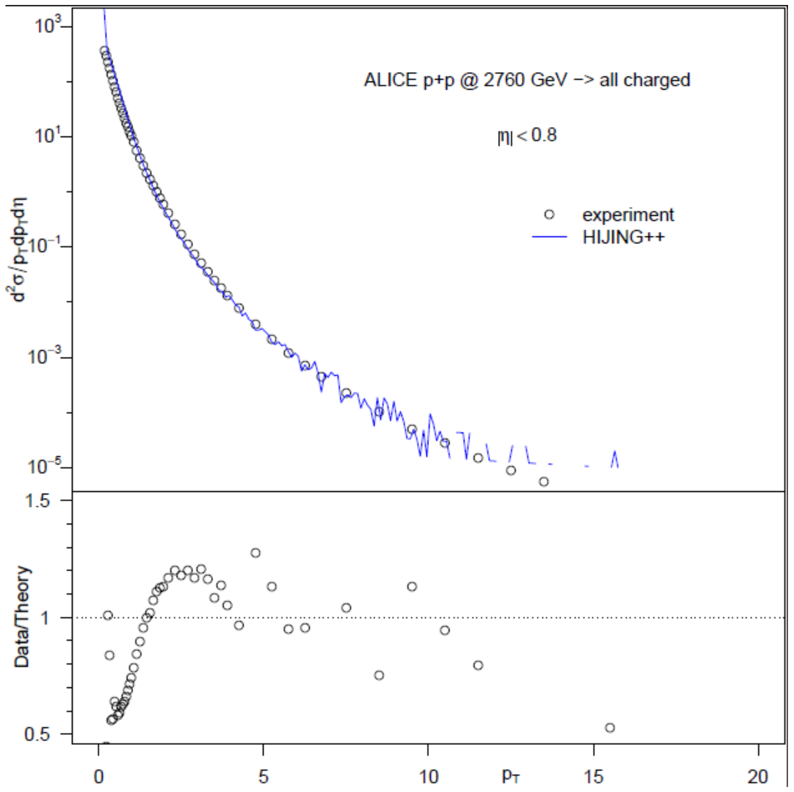


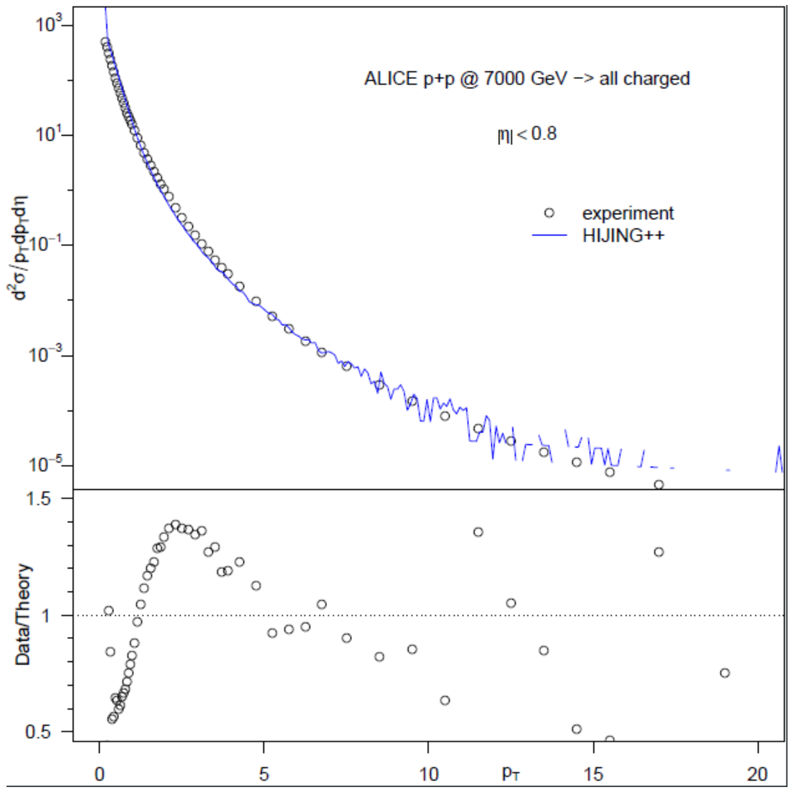
STAR Collaboration, Phys.Lett. B637 page 161-169 (2006)

Apetizer plots for the LHC era

Code validation with LHC pp data at 900, 2760, 7000 GeV c.m. energies.







ALICE Collaboration, 10^9 event

Runtime comparison

For 1e5 Events.

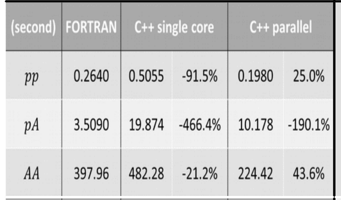
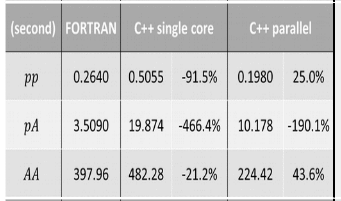
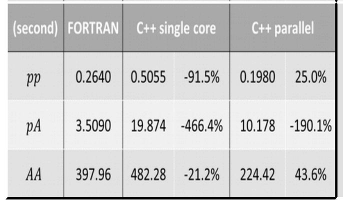
```fortran
integer::beg, end, rate
call system_clock(beg,rate)

(end - beg)/real(rate)
```

```cpp
#include <chrono>

auto start = std::chrono::high_resolution_clock::now();

double runtime = std::chrono::duration_cast<std::chrono::milliseconds>
(end.time_since_epoch() - start.time_since_epoch()).count();
```

(second)	FORTRAN	C++ single core		C++ parallel	
	0.2640	0.5055	-91.5%	0.1980	25.0%
	3.5090	19.874	-466.4%	10.178	-190.1%
	397.96	482.28	-21.2%	224.42	43.6%

(second)	FORTRAN	C++ single core		C++ parallel	
pp	0.2640	0.5055	-91.5%	0.1980	25.0%
pA	3.5090	19.874	-466.4%	10.178	-190.1%
AA	397.96	482.28	-21.2%	224.42	43.6%

Ongoing activities and future plans

- Ongoing activities (HIJING++ v3.0)
  - code/compatibility tests & tuning
  - performance test
  - new physics (Shadowing, Quenching)
  - parallel version

- Future plans (HIJINGv3.x)
  - online access – documentation
  - AliRoot compatibility
  - multi thread and GPU support
  - GUI